Analysis of the execution time of a program and the leakage of sensitive information

A Thesis
Presented to
The Academic Faculty

In Partial Fulfillment
of the Requirements for the Degree
B.S. in Computer Science with Research Option in the
College of Computing

Georgia Institute of Technology

Ami Patel
10/7/16

CHAPTER 1

INTRODUCTION

In the information security field of computer science, the questions of whether a program is safe or unsafe depends heavily on the inputs in which the program may leak sensitive information to an attacker who can observe the execution time of the program. Existing information- flow analyses achieve precision or scalability, but not both. Moreover, many of the current techniques to determine the safety of a program are slow and inefficient or require to much man power. Because of this, the inputs of programs and possible automatic research methods are being researched to fill the gap in determining whether a program is safe or unsafe.

When determining the level of security a program carries, many computer scientists have looked into how information flow control addresses privacy. However, because this method is not widely used and lacks many core functionalities, this research looked into aliasing, callbacks, arrays and lists along with possible automatic checkers using logic. The benchmarks detail information on safe inputs and outputs and separate each step into classified loops, methods, and recursion.

Information flow control and Classical static analysis enable logical and denotational techniques to determine the safeness of a program (Myers, Andrew & Liskov). However, the restrictive nature of both models strictly limits the scope for complete security (Myers, Andrew & Liskov). Therefore, the inputs of programs are being studied to help in determining whether a program is safe or unsafe. The manner in which information flow control addresses privacy has

been delved into using many different methods. However, as discussed briefly above, many of these methods are not widely used, lack many core functionalities, and are restrictive (Myers, Andrew & Liskov).

Ami Patel
10/7/16

CHAPTER 2

METHODS AND MATERIALS

In present information flow control policies, this research looked into aliasing, callbacks, arrays and lists. These benchmarks detailed information on safe inputs and outputs and separated each step into classified loops, methods, and recursion. By looking at the Java code in which each simple benchmark had been written, changing the parameters and inputting different strings, the possibility of a leak was determined by obtaining an output.

Using this input/output technique to compile and run basic loops and methods accurately helped to determine if a program was completely safe or unsafe. Each program was broken down into parts of the code that used an if/else statement, a for/ while loop or a using a try/catch statement. The ability to break Java code down into these simple functions allowed the generator to input a set of strings and in return output them as well. Running through the outputs using various algorithms enabled the detection of security leaks.

I further researched the details aliasing, arrays, lists, and callbacks by using DroidBench, a benchmark on GitHub to further research each classification. With these classifications, I studied the recursion, loops, and other basic object oriented java functions within these benchmarks. I discussed the intermediate steps to understand the process and then dove into them to form a framework; Then, I helped in creating and testing an automatic verifier to look into the inputs of each program, ultimately determining the level of security each program contains by comparing the paths of each input. The automatic verifier was implemented using the Java Virtual Machine language.

Ami Patel
10/7/16

CHAPTER 3

RESULTS

Running through the outputs using various algorithms and benchmarks enabled the detection of security leaks. This is because each and every output was traced over to make sure that complete security was being used. This was quite impressive as programs were easily and quickly run without much concern or work from the programmer's side.

To showcase this, an automatic verifier for commodity languages was presented named Symone. It selects pairs of paths and runs the pair trying to combine a proof of the flow security. Once this is done, if it finds a path of pairs that were picked before, it is able to find a general proof that all the chosen paths satisfy. Symone was implemented in the Java Virtual Machine language and was tested on a set of information flow benchmarks along with case studied from the Google Play app marketplace. It was found that Symone either verifies or invalidates each policy satisfaction for the pairs properly.

Ami Patel
10/7/16

CHAPTER 4

DISCUSSION

Although Information flow control and Classical static analysis have focused on the design, implementation, and testing of program safety, they had not focused solely on the execution time of the program and the manner in which it leaks sensitive information on an operated program. (Mcmillan). This is important because all the intricate details of a computer leakage are revealed through the inputs and outputs. The execution time and input/output characteristics of programs can give an exact number of leaks if any exist ultimately determining the safety of programs. Information flow control and Classical static analysis use optimizing transformations to enable logical and denotation techniques (Myers, Andrew & Liskov). Elimination transformation is justified with logic with these methods as well (Myers, Andrew & Liskov). Such functions allow most security checks to be monitored. Because of this, the details of certain algorithms and benchmarks were studied to determine the security by the input and output of various programs. Doing so checked for the security and the automatic verifier; Symone was able to either verify or invalidate each policy fulfillment for each pair.